



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/751,761	12/29/2000	Ronald D. Smith	2207/10119	5065
7590 Kenyon & Kenyon Suite 600 333 W. San Carlos Street San Jose, CA 95110-2711	08/06/2007		EXAMINER HUISMAN, DAVID J	
			ART UNIT 2183	PAPER NUMBER
			MAIL DATE 08/06/2007	DELIVERY MODE PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

MAILED

AUG 06 2007

Technology Center 2100

**BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES**

Application Number: 09/751,761

Filing Date: December 29, 2000

Appellant(s): SMITH, RONALD D.

Jeffrey R. Joseph, Reg. No. 54,204
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed on April 19, 2007, appealing from the Office action mailed on June 26, 2006.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct. It should be noted that the examiner did intend to have appellant's after-final amendment entered upon appeal despite the advisory action mailed on October 12, 2006, indicating otherwise. The examiner asserts that box 7(b) should have been checked instead of box 7(a) in the advisory action.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

- Swoboda et al., U.S. Patent No. 6,643,803 (November 2003)
- Ehlig et al., U.S. Patent No. 5,551,050 (August 1996)
- Sato, U.S. Patent No. 5,903,768 (May 1999)
- Mandyam et al., U.S. Patent No. 6,285,974 (September 2001)
- Hennessy and Patterson, "Computer Organization and Design, 2nd Edition, 1998

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claim Rejections - 35 USC § 103

1. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.
2. Claims 20, 24, 26, 33, and 37 are rejected under 35 U.S.C. 103(a) as being unpatentable over Swoboda et al., U.S. Patent No. 6,643,803 (as applied in the previous Office Action and herein referred to as Swoboda) in view of Ehlig et al., U.S. Patent No. 5,551,050 (herein referred to as Ehlig).
3. Referring to claim 20, Swoboda has taught a method comprising:
 - a) detecting a stall in an execution stage of a processor core. See claim 1 and the abstract, and note that a bubble/stall is detected.
 - b) generating a neutral instruction. See claim 1 and the abstract, and note that a system resource access is generated. And, the instruction is neutral because a resource can be read as a result of

the instruction. Reading a resource does not modify the architectural state of the processor, and therefore, a read is a neutral instruction.

c) providing said neutral instruction to said execution stage. See claim 1 and the abstract, and note that the system resource access is jammed into the bubble, which is in the pipeline.

d) executing said neutral instruction to ascertain an architectural state value for said processor core. See claim 1 and the abstract, and note that the access is executed and, as a result, a value is obtained.

e) Swoboda has not taught comparing said architectural state value for said processor core to an architectural state value for a second processor core. However, Ehlig has taught the concept of comparing first processor data obtained from emulation circuitry to second processor data obtained from emulation circuitry. See column 6, lines 7-22. As Ehlig explains, this type of comparison is useful in redundant processing applications which vote. As is known, redundancy and voting allows for increased fault tolerance within the system. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Swoboda such that at least one redundant processor is implemented and the results of the processors are compared. As explained, one would be motivated to make such a combination to increase fault tolerance. And, it follows that when debugging a redundant system, the outputs of the redundant processors should be compared to ensure that they are operating in redundant fashion. To not do so would be to rely too heavily on assumption that the system will provide appropriate fault tolerance.

4. Referring to claim 24, Swoboda has taught a method as described in claim 20. Swoboda has further taught that the execution of said neutral instruction causes said processor core to

access a value stored in a register in said processor core. From the abstract it is disclosed that system resources, which include registers (column 2, lines 50-51), are read when a neutral instruction is executed.

5. Referring to claim 26, Swoboda has taught a system comprising:

a) stall logic coupled to an execution stage of a processor core to detect a stall in said execution.

See claim 1 and the abstract, and note that a bubble/stall is detected.

b) comparison logic coupled to said execution stage, wherein upon occurrence of the stall said execution stage is to execute a neutral instruction to ascertain an architectural state value for said processor core. See claim 1 and the abstract, and note that, in response to a bubble/stall, a system resource access is jammed into the bubble and executed, thereby avoiding additional pipeline delay. A system resource includes registers (column 2, lines 50-51). Since registers hold values, these jammed instructions would ascertain (read) values for the processor. In addition, it should be realized that these instructions are used for testing purposes, as described in column 2, lines 46-65. Therefore, for a test to occur, comparison logic must inherently exist in order to determine whether the test was a failure or success. There must be an expected outcome of some sorts which would be compared with the outcome obtained from executing the neutral instruction.

c) Swoboda has not taught comparing said architectural state value for said processor core to an architectural state value for a second processor core. However, Ehlig has taught the concept of comparing first processor data obtained from emulation circuitry to second processor data obtained from emulation circuitry. See column 6, lines 7-22. As Ehlig explains, this type of comparison is useful in redundant processing applications which vote. As is known, redundancy

and voting allows for increased fault tolerance within the system. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Swoboda such that at least one redundant processor is implemented and the results of the processors are compared. As explained, one would be motivated to make such a combination to increase fault tolerance. And, it follows that when debugging a redundant system, the outputs of the redundant processors should be compared to ensure that they are operating in redundant fashion. To not do so would be to rely too heavily on assumption that the system will provide appropriate fault tolerance.

6. Referring to claim 33, Swoboda has taught a set of instructions residing in a storage medium (Fig.12, and note the instruction memory on the far left - "INST MEM"), said set of instructions capable of being executed in an execution stage by a processor core for implementing a method to test the processor core, the method comprising:
 - a) detecting a stall in an execution stage of a processor. See claim 1 and the abstract, and note that a bubble/stall is detected.
 - b) generating a neutral instruction. See claim 1 and the abstract, and note that a system resource access is generated. And, the instruction is neutral because a resource can be read as a result of the instruction. Reading a resource does not modify the architectural state of the processor, and therefore, a read is a neutral instruction.
 - c) providing said neutral instruction to said execution stage. See claim 1 and the abstract, and note that the system resource access is jammed into the bubble, which is in the pipeline.

d) executing said neutral instruction to ascertain an architectural state value for said processor core. See claim 1 and the abstract, and note that the access is executed and as a result, a value is obtained.

e) Swoboda has not taught comparing said architectural state value for said processor core to an architectural state value for a second processor core. However, Ehlig has taught the concept of comparing first processor data obtained from emulation circuitry to second processor data obtained from emulation circuitry. See column 6, lines 7-22. As Ehlig explains, this type of comparison is useful in redundant processing applications which vote. As is known, redundancy and voting allows for increased fault tolerance within the system. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Swoboda such that at least one redundant processor is implemented and the results of the processors are compared. As explained, one would be motivated to make such a combination to increase fault tolerance. And, it follows that when debugging a redundant system, the outputs of the redundant processors should be compared to ensure that they are operating in redundant fashion. To not do so would be to rely too heavily on assumption that the system will provide appropriate fault tolerance.

7. Referring to claim 37, Swoboda has taught a set of instructions as described in claim 33. Swoboda has further taught that in said method the execution of said neutral instruction causes said processor core to access a value stored in a register in said processor core. From the abstract it is disclosed that system resources, which include registers (column 2, lines 50-51), are read when a neutral instruction is executed.

8. Claims 21, 27, and 34 are rejected under 35 U.S.C. 103(a) as being unpatentable over Swoboda in view of Ehlig and further in view of Sato, U.S. Patent No. 5,903,768 (as applied in the previous Office Action).

9. Referring to claim 21, Swoboda in view of Ehlig has taught a method as described in claim 20. Swoboda in view of Ehlig has not taught that said neutral instruction is generated when a plurality of instructions are generated by a compiler. However, Sato has taught such a concept. More specifically, in column 2, lines 4-13, Sato discloses that a compiler is used to generate instructions and the order in which they are executed. Furthermore, when a hazard between two instructions cannot be eliminated, the compiler inserts a NOP instruction between them to overcome the hazard. This is equivalent to generating a neutral instruction because the neutral instruction is the same as a NOP in the sense that it does not affect the architectural state of the processor. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Swoboda in view of Ehlig such that a neutral instruction is generated when a plurality of instructions are generated by the compiler. This would be obvious because Sato has taught the known concept of overcoming a hazard by generating NOPs (or neutral instructions), and hazards must be overcome in order to prevent data corruption.

10. Referring to claim 27, Swoboda in view of Ehlig has taught a system as described in claim 26. Furthermore, claim 27 is rejected for the same reasons set forth in claim 21.

11. Referring to claim 34, Swoboda in view of Ehlig has taught a set of instructions as described in claim 33. Furthermore, claim 34 is rejected for the same reasons set forth in claim 21.

12. Claims 22-23, 25, 28-30, 35-36, and 38 are rejected under 35 U.S.C. 103(a) as being unpatentable over Swoboda in view of Ehlig and further in view of Mandyam et al., U.S. Patent No. 6,285,974 (as applied in the previous Office Action and herein referred to as Mandyam).

13. Referring to claim 22, Swoboda in view of Ehlig has taught a method as described in claim 20. Swoboda in view of Ehlig has not taught that said neutral instruction is generated by a No-operation (NOP) pseudo-random generator. However, Mandyam has taught generating test instructions using a random test generator. A person of ordinary skill in the art would have recognized that by implementing a random generator to generate instructions, sources of bias are eliminated. Consequently, truly random instructions may be generated which would allow for the possibility of testing any register at any appropriate point within the execution. As a result, in order to perform random testing, as opposed to biased testing, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Swoboda in view of Ehlig such that a No-operation (neutral instruction) pseudo-random generator is used to generate neutral instructions, as taught by Mandyam.

14. Referring to claim 23, Swoboda in view of Ehlig and further in view of Mandyam has taught a method as described in claim 22. Swoboda has further taught that the execution of said neutral instruction causes said processor core to access a value stored in a register in said processor core. From the abstract it is disclosed that system resources, which include registers (column 2, lines 50-51), are read when a neutral instruction is executed.

15. Referring to claim 25, Swoboda in view of Ehlig has taught a method as described in claim 20. Swoboda in view of Ehlig has not taught that said neutral instruction is generated by a post-processor device. However, Mandyam has taught such a concept. See Fig.3 and column 6,

lines 10-14. Note that a post-processor is used to generate instructions which are used to perform a self-check. This allows for detection of architectural violations as described in column 6, lines 10-25. Since Mandyam has taught that test instructions may be generated by a post-processor, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement a post-processor in Swoboda in view of Ehlig for such a purpose.

16. Referring to claim 28, Swoboda in view of Ehlig has taught a system as described in claim 26. Furthermore, claim 28 is rejected for the same reasons set forth in claim 22.

17. Referring to claim 29, Swoboda in view of Ehlig has taught a system as described in claim 28. Furthermore, claim 29 is rejected for the same reasons set forth in claim 23.

18. Referring to claim 30, Swoboda in view of Ehlig and further in view of Mandyam has taught a system as described in claim 29. Swoboda in view of Ehlig and further in view of Mandyam has not taught that said neutral instruction includes ORing the contents of said register with itself. However, an OR operation is well known and expected in the art. And, it is known that ORing an operand with itself is a neutral operation as ORing 0 and 0 yields 0 and ORing 1 and 1 yields 1. Since an OR operation is a fundamental logic operation, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement the function of ORing the contents of a register with itself.

19. Referring to claim 35, Swoboda in view of Ehlig has taught a set of instructions as described in claim 33. Furthermore, claim 35 is rejected for the same reasons set forth in claim 22.

20. Referring to claim 36, Swoboda in view of Ehlig has taught a set of instructions as described in claim 35. Furthermore, claim 36 is rejected for the same reasons set forth in claim 23.

21. Referring to claim 38, Swoboda in view of Ehlig has taught a set of instructions as described in claim 33. Furthermore, claim 38 is rejected for the same reasons set forth in claim 25.

22. Claims 31-32 are rejected under 35 U.S.C. 103(a) as being unpatentable over Swoboda in view of Ehlig in view of Mandyam and further in view of Hennessy and Patterson, Computer Organization and Design, 2nd Edition, 1998 (as applied in the previous Office Action and herein referred to as Hennessy).

23. Referring to claim 31, Swoboda in view of Ehlig and further in view of Mandyam has taught a system as described in claim 29. Swoboda in view of Ehlig and further in view of Mandyam has not taught that said neutral instruction includes ANDing the contents of said register with all binary 1 values. However, an AND operation is well known and expected in the art, and supported by Hennessy. See pages 225-226. Hennessy has taught that each resulting bit will be 1 only if both corresponding operand bits are 1. And, an operand of an AND operation can be an operand of all 0's, an operand of all 1's, and everything in between. Masking (ANDing operation) is used to isolate fields, which in turn allows for the examination of bits within a word. Consequently, since an AND operation is a fundamental logic operation, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement the function of ANDing the contents of a register with all binary 1 values, as taught by Hennessy.

24. Referring to claim 32, Swoboda in view of Ehlig and further in view of Mandyam has taught a system as described in claim 29. Swoboda in view of Ehlig and further in view of Mandyam has not taught that said neutral instruction includes ORing the contents of said register with all binary 0 values. However, an OR operation is well known and expected in the art, and supported by Hennessy. See pages 225 and 227. Hennessy has taught that each resulting bit will be 1 if either one of the corresponding operand bits are 1. And, an operand of an OR operation can be an operand of all 0's, an operand of all 1's, and everything in between. Consequently, since an OR operation is a fundamental logic operation, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement the function of ORing the contents of a register with all binary 0 values, as taught by Hennessy.

(10) Response to Argument

Regarding Argument A(i) on pages 5-6 of the appeal brief:

- Appellant argues on page 5, that:

"...it is clear from appellant's claim that the neutral instruction is being executed by the processor core...the read instruction in Swoboda is not being executed by the processor. Instead, unlike in appellant's claim, the instruction is being executed by the emulation circuitry, which is separate from the processor."

This argument has been considered and deemed non-persuasive for the following reasons:

a) Multiple passages in Swoboda point to the processor executing the neutral instruction and not to the emulation circuitry executing the neutral instruction, as argued by appellant.

First, see the abstract of Swoboda, and note that the emulation circuitry (Fig.14, component 851) jams an instruction into the instruction register of the processor (Fig.14,

component 102). Clearly, this passage alone strongly supports the examiner's rejection. An instruction register (IR) is a known processor component that stores an instruction to be executed. The teaching of this component is found in many beginner computer architecture textbooks. As is known, jamming/storing an instruction into the processor's IR causes the processor to execute that instruction. In addition, claim 1 of Swoboda states that the processor comprises an instruction pipeline in which a system resource access operation (neutral instruction) is jammed.

Secondly, the abstract of Swoboda states that the jammed instruction causes resources to be read on behalf of the emulation circuitry. Why would the read operation be performed on behalf of the emulation circuitry if the emulation circuitry were itself executing the instruction? This is further evidence that the CPU (processor core) executes the neutral instruction.

Furthermore, see column 25, lines 42-57, and note that the CPU (processor core) is the component that executes the read instruction before passing the read data to the DT_DMA (emulation circuitry, Fig.15A, component 851). Hence the processor executes the neutral instruction to perform a read for the emulation circuitry.

Finally, even if the above passages didn't support the examiner, though they clearly do, the examiner asserts that an alternate interpretation of Swoboda would include at least the CPU 102 and emulation circuitry 851 of Fig.14 being collectively referred to as a "processor" as both of these components collectively process data. Swoboda even supports this interpretation, which is apparent from Swoboda's claim 1, in which the processor comprises an instruction pipeline in which a neutral instruction is jammed and the emulation circuitry for debug events. Consequently, whether the emulation circuitry executes the neutral instruction, which appellant

contends and examiner disagrees with, or the processor core does the execution, as supported above by the examiner, the “processor” does the execution, where the processor is interpreted as either the CPU component 102 of Fig.14 or as a unit including at least the CPU 102 and emulation circuitry 851 of Fig.14.

In summation, the examiner has certainly provided adequate evidence and support for the processor core of Swoboda executing the neutral instruction. The emulation circuitry does not execute the neutral instruction as appellant claims. Consequently, the examiner should be affirmed on this topic.

- Appellant argues on page 6, that:

“...Swoboda does not teach “executing said neutral instruction to ascertain an architectural state value for said processor core.” On page 3 at section 6d, the Office Action asserts that claim 1 and the abstract of Swoboda teach this element, but appellant cannot find anywhere in the cited sections of Swoboda where ascertaining an architectural state value is taught. The office action states that the cited sections of Swoboda teach receiving “a value,” but the Office Action does not show what the value is. It appears Swoboda is referring to receiving instructions, not architectural state values of processor cores. The Office Action also does not show that this undefined “value” is being received as the result of executing a neutral instruction.”

This argument has been considered and deemed non-persuasive for the following reasons:

The examiner feels that this limitation is also clearly taught. As is known, debugging is the methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of hardware. During debugging, the user accesses architectural values of the system in order to determine if the expected results are being produced. If unexpected results are obtained, then it is determined that an error exists and, subsequently, the user would correct it such that expected results are obtained in the future. The abstract of Swoboda clearly refers to

debugging, and it is disclosed that “the emulation circuitry can jam an instruction into the instruction register of the processor to cause processor resources to be read or written...” Swoboda further expands on this concept in column 2, lines 50-51, and column 3, lines 13-25, by stating that the processor resources that are read are registers. Since Swoboda is concerned with debugging, and therefore needs a way to be able to access architectural values to pinpoint errors, Swoboda provides a neutral instruction that, when jammed into the processor, causes a processor resource (i.e., register) to be read and the value in the register to be obtained. The value read from a register is an architectural state value. Note that a read instruction is a neutral instruction because a read has no changing effect on the state of the system. That is, unlike a write instruction, for example, which writes data to a memory location, and consequently changes the state of the system, a read instruction simply reads a value without changing anything. Hence, a read instruction is a neutral instruction.

Appellant argues that Swoboda is not receiving such a value, but is instead receiving instructions. The examiner asserts that, while Swoboda’s processor does receive neutral instructions, this is not the received architectural state value. It is the neutral instruction that, when executed, causes an architectural state value to be obtained. And this is all done to support debugging.

In summation, the examiner has certainly provided adequate evidence and support for the executing the neutral instruction by the processor core to ascertain an architectural state value of the core. Appellant is incorrect in arguing that Swoboda has not taught ascertaining an architectural state value and that the value is “undefined”. Consequently, the examiner should be affirmed on this topic.

Regarding Argument A(ii) on pages 6-7 of the appeal brief:

- Appellant argues on page 6, that:

"Page 3, section 6e, of the Office Action dated June 26, 2006 admits that Swoboda does not teach comparing the architectural state value for a processor core to an architectural state value for a second processor core. The Office Action claims this deficiency is made up for in Ehlig, which according to the Office Action, teaches "the concept of comparing first processor data obtained from emulation circuitry to second processor data obtained from emulation circuitry." The Examiner, however, has not shown that the processor data in Ehlig represents an architectural state value of a processor core."

This argument has been considered and deemed non-persuasive for the following reasons:

Ehlig has specifically disclosed, in column 6, lines 7-10, processors that calculate the same information and vote on the answer. That is, in order to increase fault tolerance, each processor performs the same computation and the results are compared. If each answer is equivalent, then probability suggests that the answer is almost certainly correct. Non-matching results indicate that a fault has occurred in at least one of the processors but the fault would be masked if other processors agree on the correct result. For example, if a system had three redundant processors, thereby forming a triple modular redundancy system, then each processor would perform a same operation and produce a result. Each result is analyzed by a voting mechanism that compares the results and chooses the majority result as the final result. For instance, assume two processors produce the correct result, and the third processor produces an incorrect result. Because the correct result is the majority result, it is consequently and correctly chosen as the final result. If a single processor system produces a fault, there is no way of "hiding" the fault. It is for this reason that redundant voting schemes are useful and accepted in the art. However, in response to this particular argument that architectural values are not

compared in Ehlig, the examiner asserts that the result (information) that is calculated and voted upon is an architectural state value for at least two reasons:

1) Broadly interpreted, an architectural state value is merely a value produced by the processing architecture. It should be noted that the examiner has been unable to find any explicit definition in the specification of an architectural state value, and consequently, the examiner is to give the broadest reasonable interpretation to architectural state value (*"An examiner has the duty to police claim language by giving it the broadest reasonable interpretation."* *Springs Window Fashions LP v. Novo Industries, L.P.*, 65 USPQ2d 1826, 1830 (Fed. Cir. 2003)). Therefore, an architectural state value is simply a value produced by a processing device, and Ehlig has clearly taught such a value.

2) Applying a more narrow interpretation, an architectural state value is sometimes viewed as a “true” value that is ultimately committed to a register, which modern processors clearly include. The opposite of an architectural value is a speculative value, which is a value generated by a speculatively executed instruction (i.e., an instruction that is executed without fully knowing if that instruction should be executed). So, a speculative value is a value that is obtained but will ultimately be discarded if the value was produced by an instruction that executed speculatively but should not have executed (such is the case with mispredicting a branch instruction). The details of speculative execution are not particularly important for this response, but what is important is that Ehlig makes no reference to any type of speculative execution or equivalent thereof. Therefore, speculative values do not exist in Ehlig, and consequently, all values in Ehlig are values which will be committed to registers (architectural state values).

In summation, despite applying the broadest reasonable interpretation or even a more narrow interpretation of “architectural state value”, the examiner asserts that the values produced and compared in the processing architecture of Ehlig are architectural values. Appellant is mistaken in arguing that the examiner has not shown that the processor data in Ehlig represents an architectural state value of a processor core. Consequently, the examiner should be affirmed on this topic.

- Appellant argues on page 7, that:

“Ehlig teaches synchronizing redundant processors so that data output to memory for the multiple processors can be compared, thus increasing the fault tolerance by having the processors vote. The examiner states that it would have been obvious to one of ordinary skill in the art to add a redundant processor to the disclosure of Swoboda, but the abstract of Swoboda states that the invention described is optimized for devices such as wireless telephones. Size, power consumption, cost, and many other factors make adding redundant processors to the invention described in Swoboda impractical.”

This argument has been considered and deemed non-persuasive for the following reasons:

First, the examiner will summarize the combination and reason for making it. Swoboda is concerned with debugging uniprocessor and multiprocessor systems. For instance, see Fig.14 and note the single CPU, and see Fig.2, and column 5, lines 57-62, and note the debugging of multiple processors. The nature of the multiprocessor system is unknown. Are they two different processors? Or, could they conceivably be redundant processors used to improve fault tolerance as described below? Ehlig shows redundant processors used in a typical voting scheme to improve fault tolerance. That is, integrated circuits, in general, are frequently used in the presence of radiation such as x-rays, gamma-rays, photons, particles, etc. A radiation strike can deposit charge in silicon and, therefore, can cause upsets in the integrated circuits. As a result of

such radiation, charges can be collected at circuit nodes that send the nodes to unintended opposite voltage states (e.g., from high to low). When this voltage state change happens to a data storage circuit, for example, the data storage nodes change to the wrong data states. If one envisions using Swoboda's multiprocessor system of Fig.2 in any environment which is exposed to radiation, which is clearly covered by any one of the wireless telephone or control applications disclosed in Swoboda's abstract, then one would recognize that in order to better protect against radiation faults, it would have been obvious to modify Swoboda in view of Ehlig to have the multiprocessor system be a redundant processing system whose individual outputs are compared to ensure higher tolerance of faults. Similarly, adding a redundant processor to Swoboda's uniprocessor system is obvious for the same reasons. The general idea is that Swoboda in view of Ehlig, as modified, is now able to debug a redundant multiprocessor system. Clearly, one of ordinary skill in the art would have recognized that debugging and verifying that such a system works is necessary, as one would not employ such a processor to control an expensive satellite in space, for example, without first testing to see if it would withstand radiation faults more commonly found in space. Since the idea of a redundant system is to have individual processors perform the same task and produce the same result (as a form of verification), then to ensure that the individual processors are performing identically, values from each processor should be read, via neutral instruction as taught by Swoboda, and compared to ensure that they match, and to in turn ensure that the redundancy is working. This is the idea behind the combination: to produce a debugging environment for a redundant multiproessor system.

Regarding appellant's argument that such a combination would be impractical in terms of size, power, cost, etc. because Swoboda is concerned with wireless telephones, the examiner

asserts that appellant has incorrectly limited Swoboda's invention to wireless telephones, which is not the case. Swoboda's abstract states that the architecture and instruction set are optimized for high efficiency execution of algorithms for wireless telephones and pure control tasks. Clearly, items in higher radiation environments need to be controlled (for instance, satellites in space), and Swoboda's processor is useful and intended for control applications. That is, the system of Swoboda is not limited to wireless technology but is instead intended for any controller that employs a control algorithm. Furthermore, Swoboda's abstract's listing of wireless telephone control is simply exemplary. It cannot be concluded by appellant that Swoboda cannot be modified by Ehlig simply because adding at least one redundant processor would be impractical in wireless applications. Anything that can be controlled can make use of Swoboda's and Ehlig's invention.

In addition, appellant argues that multiple processors or adding a redundant processor is impractical in Swoboda, yet Swoboda has already taught a multiprocessor system (see Fig.2 and column 5, lines 57-62). A redundant processor system is merely a multiprocessor system in which each individual processor performs the same calculation. So, it cannot be concluded that Swoboda cannot be modified to include multiple processors (in which one is redundant) due to size, power, and cost constraints, when Swoboda has already essentially taught the required hardware (multiple processors). The combination, in the very least, would require making one of the processors shown in Fig.2 of Swoboda a redundant processor.

In summation, the examiner has certainly provided valid reasons for combining Swoboda and Ehlig and has also provided valid reasons why it is not impractical in terms of size, cost, power, etc. and that the combination is not limited to wireless telephones (as argued by

Art Unit: 2144

appellant), but to any control application (where clearly control applications need to be executed in environments more susceptible to radiation faults). Consequently, the examiner should be affirmed on this topic.

- Appellant argues on page 7, that:

"Additionally, the manual debugging method taught by Swoboda is not readily compatible with a redundant processor system like that in Ehlig. The debugging method in Swoboda allows for a user to manually "fix" problems that processors encounter. The processors in Ehlig are all synchronous. If one processor needs to enter a debug state, as described in Swoboda, then all the other processors would be forced to do the same. Forcing all the processors to stop because of an error on one processor would defeat one of the primary purposes of having a redundant processor system in the first place."

This argument has been considered and deemed non-persuasive for the following reasons:

a) The examiner does not really understand how this argument is relevant. As mentioned above, redundant processors should clearly be debugged as well. An untested redundant processing system will not just be implemented in an expensive system (for instance, to control a satellite in space). Instead, it must first be determined that the individual processors of the system are able to produce the same results. During debugging of Swoboda in view of Ehlig's redundant system, the results of the cores must be compared so that it can be determined that the redundant system is working properly before deploying it. To not perform such a comparison would mean that the debuggers are assuming that the redundant system would work without testing it. This is clearly undesirable and erroneous. Appellant's arguments about it being undesirable to stop all of the processors during debugging does not seem to hold water because in order to determine that each processor is working the same on the same set of inputs, each processor must be stopped and checked against one another. Appellant appears to be arguing about performance when outside

of the debugging environment. Sure, if one redundant processor stops processing for whatever reason while in its actual operating environment, then it is not desirable for all processor to stop. However, this is only true when the redundant processor is outside of development stage and actually in its desired environment performing the desired application. When in the debugging/development environment, the only way to determine if each processor is performing the same at all times is to stop each processor when a result is to be checked and compare the results of each one.

In summation, the examiner has certainly provided valid reasons for combining Swoboda and Ehlig. The examiner also asserts that appellant is arguing why the combination is undesirable based on an environment that Swoboda and Ehlig are not immediately concerned about. That is, appellant thinks that it is counter-productive to stop all processors when one is stopped, and while this is the case after the redundant processing system has been deployed, this is not the case during development, and that is what concerns Swoboda and Ehlig. Consequently, the examiner should be affirmed on this topic.

Regarding Arguments B, C, and D on page 8 of the appeal brief:

For each of arguments B, C, and D, appellant requests that the rejection of dependent claims be reversed because they depend on allowable independent claims. However, for the arguments set forth above by the examiner, the independent claims are not allowable, and consequently, the dependent claim rejections should not be reversed.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

DJH
David J. Huisman
July 9, 2007

Conferees:

/Lynne H Browne/
Lynne H Browne
Appeal Practice Specialist, TQAS
Technology Center 2100

Eddie Chan
EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100